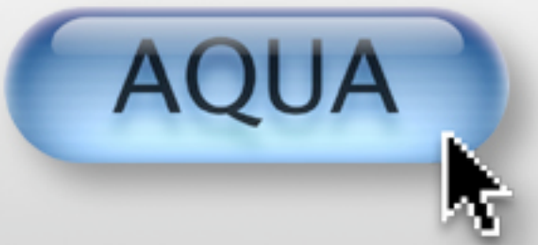
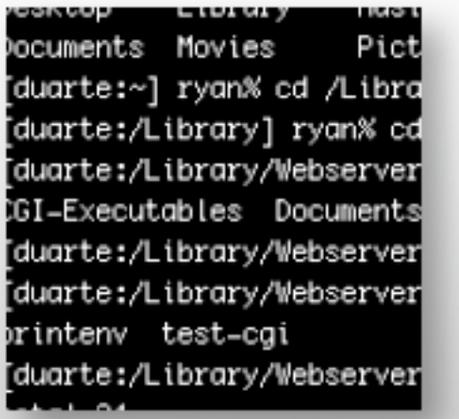
Cocoa User Interface Programming In Depth

Session 419





Cocoa User Interface Programming In Depth

Mark Piccirelli
Software Engineer, Cocoa Frameworks Group

Introduction

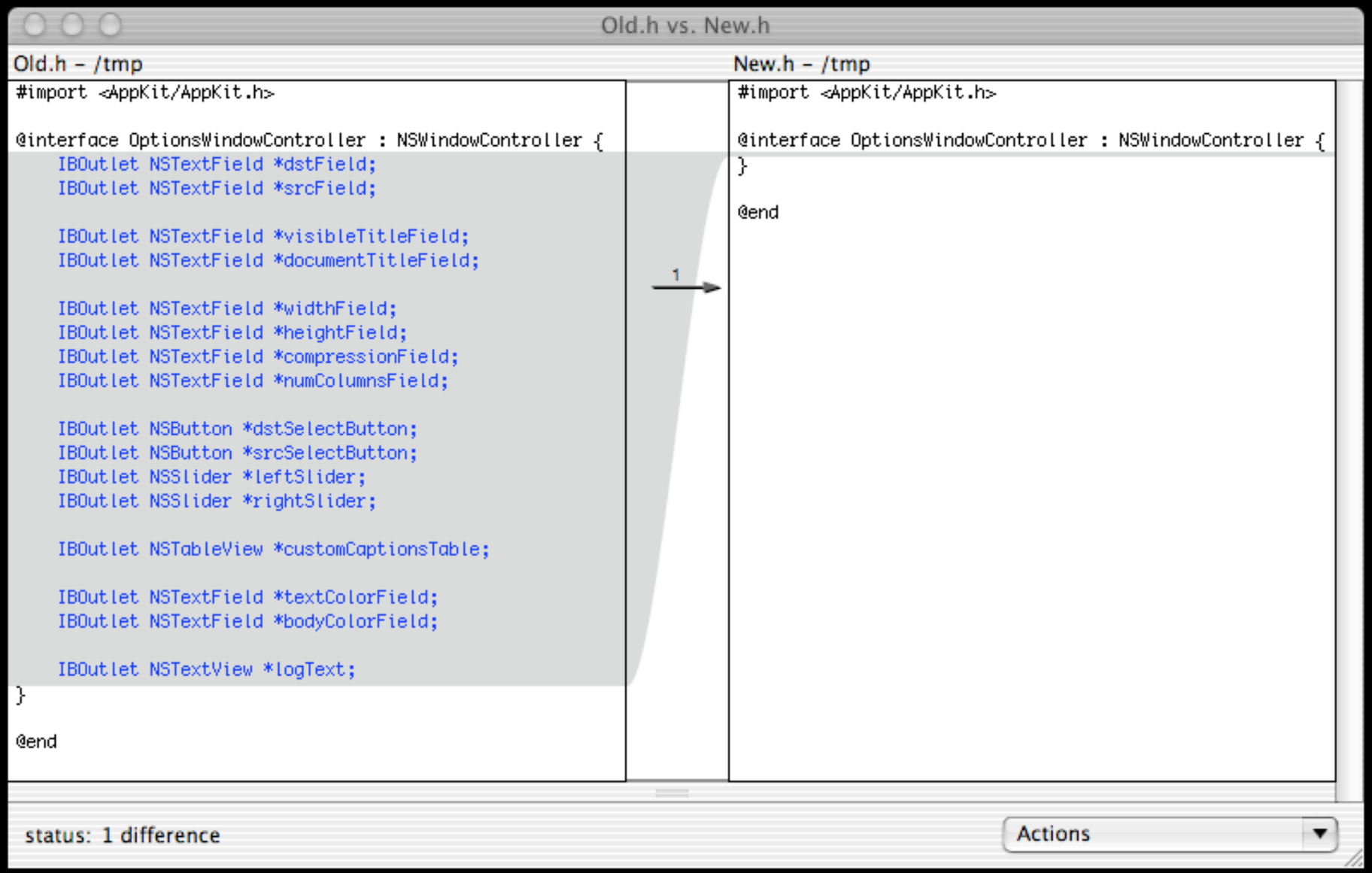
- In Session 412 on Wednesday we presented technology that will make Cocoa programming even easier
- Now let’s talk about the details

Session 412 Recap

- In Panther we are going to introduce a complete new layer of functionality—a controller layer
  - Automates user interface handling
  - Many features you have been asking for
  - Interface Builder extended to support new features without writing code
  - Makes your development faster and easier

Session 412 Recap

- Effect on your applications: less glue code

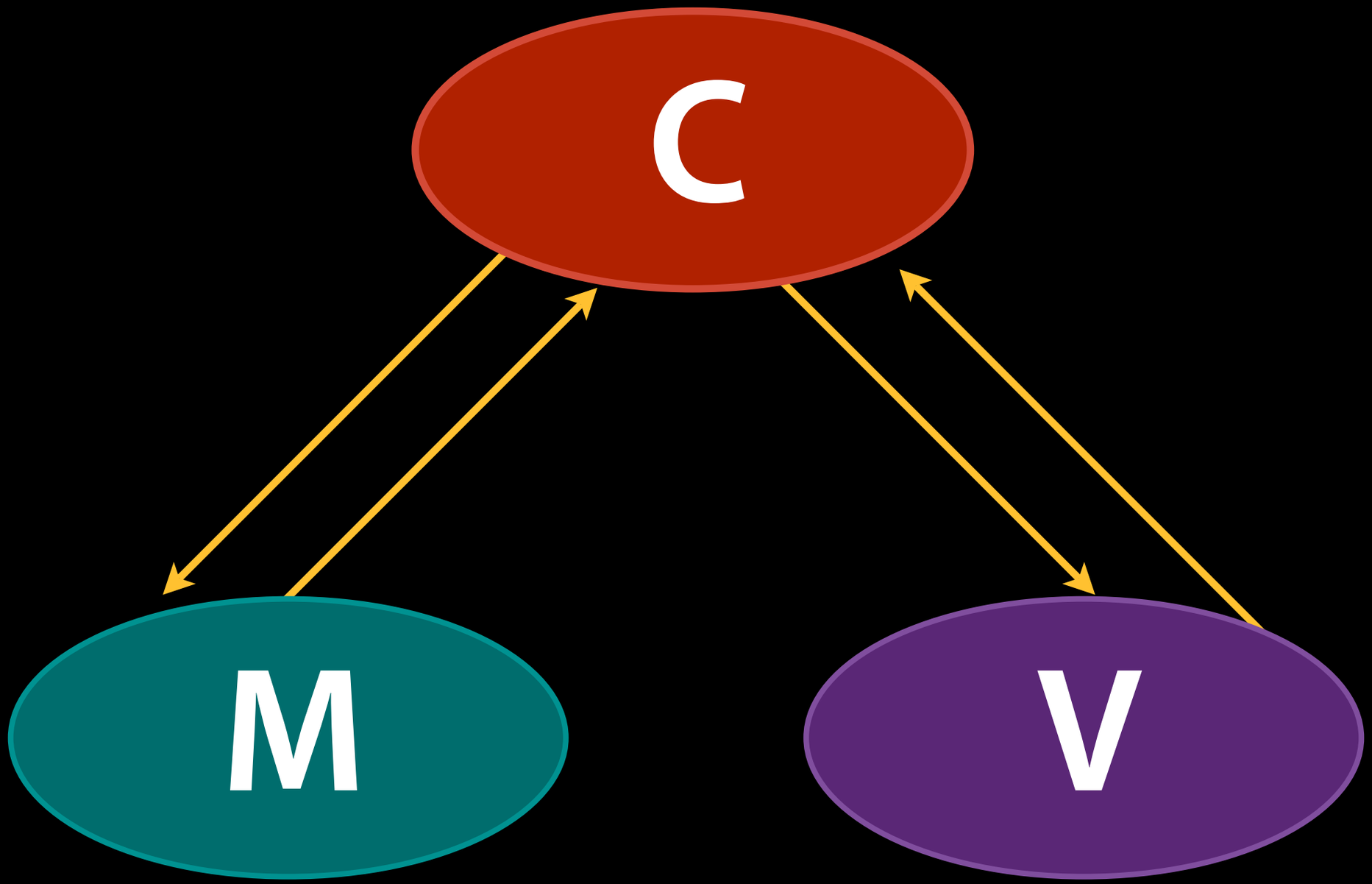


What You'll Learn

- Key Value Coding
- Key Value Observing
- Key Value Binding
- Controller Classes
- Performance

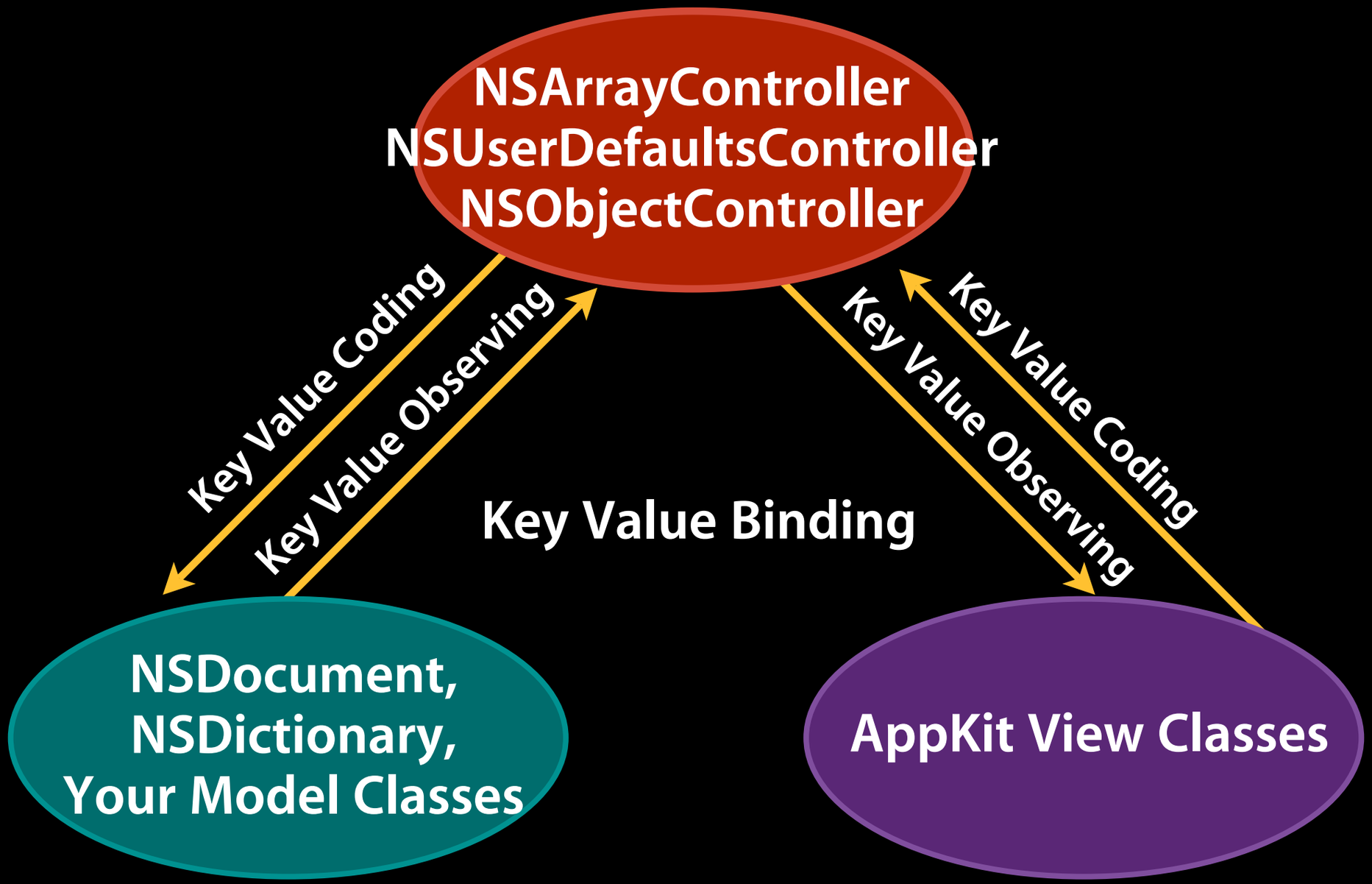
Architecture

Model-View-Controller Paradigm

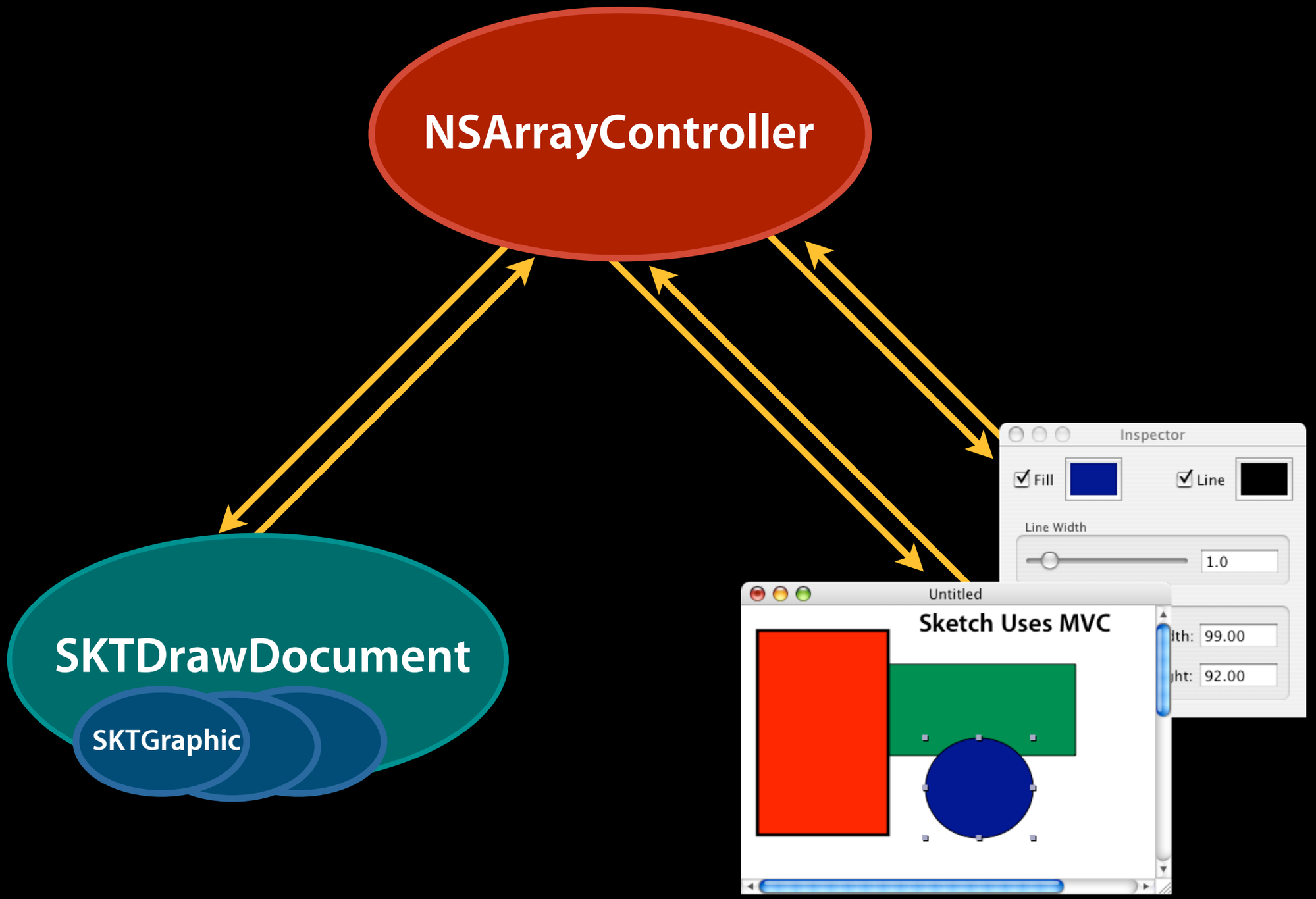


Architecture

Model-View-Controller Paradigm



Today's Example: Sketch



Terminology

- Property
  - An attribute, to-one relationship, or to-many relationship
- Attribute
  - Simple immutable value
  - E.g., the fill color or stroke width of a Sketch graphic
- To-one related object
  - Has properties of its own
  - E.g., the image of a Sketch image graphic
- To-many related objects
  - Collection of related objects
  - E.g., graphics of a Sketch document

Key Value Coding (KVC)

- Changes in existing API
- New API
- Your model objects have to be KVC-compliant for other objects to bind to them
- How to write a KVC-compliant class

Key Value Coding

What does it do?

- Gives access to object properties using a very generic API that identifies them by *key*
- Keys are just strings: “graphics,” “fillColor,” etc.
- Keys can be typed into Interface Builder’s bindings inspector
- Lets views get and set values in controllers
- Lets controllers get and set values in model objects

Key Value Coding
–[NSObject valueForKey:]

- Returns the value of the keyed property
- Default implementation searches the class for a method of the same name, and invokes it (in the simplest case)

```
[graphic valueForKey:@”fillColor”] ⇒
  [graphic fillColor]
```

Key Value Coding
–[NSObject valueForKey:]

- Converts scalar return values to NSNumbers

```
@implementation SKTGraphic
- (BOOL)drawsStroke {
  return _drawsStroke; // a boolean value
}

@end

NSNumber *drawsStroke =
  [graphic valueForKey:@”drawsStroke”];
```

Key Value Coding
–[NSObject valueForKey:]

- Now works for to-many relationships where the related objects aren't stored in an NSArray
- You could always implement

```
@implementation SKTDrawDocument
- (NSArray *)graphics { return ...; }
@end
```

- Now you can instead implement

```
- (unsigned int)countOfGraphics { return ...; }
- (SKTGraphic *)objectInGraphicsAtIndex:
    (unsigned int)index { return ...; }
```

Key Value Coding
–[NSObject valueForKey:]

- Code that uses -valueForKey: can’t tell the difference

```
NSArray *graphics = [document valueForKey:@”graphics”];
```

Key Value Coding
–[NSObject valueForKey:]

- Now supports Cocoa coding convention for boolean attributes

```
[imageGraphic valueForKey:@”flippedHorizontally”] ⇒
  [imageGraphic isFlippedHorizontally]
```

Key Value Coding
–[NSObject setValue:ForKey:]

- Sets the value of the keyed property
- Renaming of –takeValue:ForKey:
- Default implementation searches the class for a method whose name matches, and invokes it (in the simplest case)

```
[sketchGraphic setValue:aColor forKey:@”fillColor”] ⇒
  [sketchGraphic setFillColor:aColor]
```

- Converts NSNumbers to scalar values if necessary

Key Value Coding
–[NSObject validateValue:forKey:error:]

- Either returns a value good enough for –setValue:forKey: or returns an error
- Separates validation rules from decisions about when to validate
- *How* to validate is a model decision, *when* to validate is a UI decision
- Follows the typical KVC pattern; you implement

```
- (BOOL)validateZipCode:(id *)zipCode
    error:(NSError **error) {
  // Fix up the zip code and return it in zipCode and
      return YES, or return NO after setting the error.
}
```

Key Value Coding

–[NSObject mutableArrayValueForKey:]

- Returns an NSMutableArray of to-many related objects
- Use this instead of misusing -valueForKey:
- Invoker gets to send NSMutableArray messages to the result

Key Value Coding
–[NSObject mutableArrayValueForKey:]

- Model class just has to implement two methods

```
@implementation SKTDrawDocument

- (void)insertObject:(SKTGraphic *)graphic
    inGraphicsAtIndex:(unsigned int)index { ... }

- (void)removeObjectFromGraphicsAtIndex:
    (unsigned int)index { ... }

@end
```

Key Value Coding

Key paths

- For each KVC method there's a variant that takes a *key path* to access properties of related objects

```
-valueForKeyPath:
-setValue:forKeyPath:
-validateValue:forKeyPath:error:
-mutableArrayValueForKeyPath:
```

Key Value Coding

Key paths

- Dot-separated list of keys gets handled automatically

```
NSNumber *yesAsANumber = [NSNumber numberWithBool:YES];

[imageGraphic setValue:yesAsANumber
  forKeyPath:@"image.flipped"] ⇒

[[imageGraphic valueForKey:@"image"]
    setValue:yesAsANumber forKey:@"flipped"] ⇒

[[imageGraphic image] setFlipped:YES]
```

Key Value Coding

Key paths

- Key Value Binding is in terms of key paths
- Key Value Observing too
- Our controllers make use of key paths

```
[graphicsArrayController setValue:aColor
  forKeyPath:@”selection.fillColor”];
```

- You don’t have to do anything specific to support them

Key Value Coding

Making your classes KVC-compliant

- Implement enough methods for –valueForKey:, –setValue:forKey:, etc. to work
- For each attribute or to-one relationship:
  - Implement methods named –*key* and, if read-write, –set*Key*:
  - –set*Key*: shouldn't do validation
  - Implement –validate*Key*:error: if validation is appropriate

Key Value Coding

Making your classes KVC-compliant

- For each to-many relationship:
  - Implement a method named –*key*, or
  - –countOf*Key* and –objectIn*Key*AtIndex:
  - Whichever best fits your needs; invokers of –valueForKey: won't be able to tell the difference
- For each mutable to-many relationship:
  - Also implement –insertObject:in*Key*AtIndex: and –removeObjectFrom*Key*AtIndex:

Key Value Coding

NSDictionary

- Built-in KVC-compliance
- An easy way to start prototyping
- Very often good enough to ship
- No validation though

Key Value Observing (KVO)

- New API
- How to observe
- Your model objects have to be KVO-compliant (observable) for other objects to bind to them
- How to be KVO-compliant
  - Using manual observer notification
  - Using automatic observer notification

Key Value Observing

What does it do?

- Allows objects to be notified of changes to specific properties of other objects
- Lets views be notified of changes to values in controllers
- Lets controllers be notified of changes to values in model objects

Key Value Observer Registration

–[NSObject addObserver:forKeyPath:options:context:]

- Registers one object as an observer of a property of another object
- Default implementation does it all
- Receiver is the object containing the property to be observed
- Receiver must be *KVO-compliant* for the key path
- Doesn’t retain the observed object or the observer
- Use –removeObserver:forKeyPath: to deregister the observer

Key Value Observer Registration

–[NSObject addObserver:forKeyPath:options:context:]

- **observer** (NSObject *)
  - The recipient of notifications about changes to the property identified by the key path
  - Must implement the NSKeyValueObserving informal protocol

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(NSObject *)observedObject
    change:(NSDictionary *)change
    context:(void *)context;
```

  - For example, the controls in Sketch's inspector panel observe the selection properties to which they're bound

Key Value Observer Registration

–[NSObject addObserver:forKeyPath:options:context:]

- **keyPath** (NSString *)
  - The key path for the property to be observed
  - For example, “selection.fillColor”
  - Can identify an attribute, to-one relationship, or to-many relationship

Key Value Observer Registration

–[NSObject addObserver:forKeyPath:options:context:]

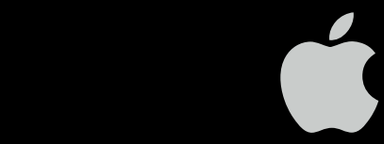
- **options** (NSKeyValueObservingOptions)
  - Any combination of two possible flags

```
NSKeyValueObservingOptionNew
```

Observer gets the new value for the property in the change notification

```
NSKeyValueObservingOptionOld
```

Observer gets the old value from *before* the change



Key Value Observer Registration

–[NSObject addObserver:forKeyPath:options:context:]

- **context** (void *)
  - Passed back in NSKeyValueObserving messages
  - Should be unique enough to identify reason for observation

Key Value Observing

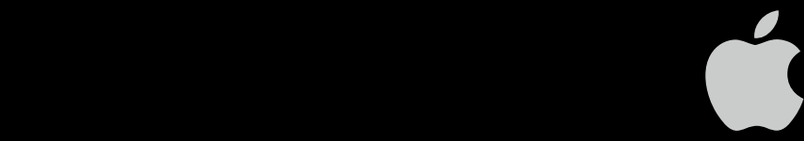
–[NSObject observeValueForKeyPath:ofObject: change:context:]

- Sent to the observer for every change to the observed property
- Makes no assumptions about what the observer actually does with the information

Key Value Observing

–[NSObject observeValueForKeyPath:ofObject: change:context:]

- **keyPath** (NSString *)—identical to the one passed in the –addObserver…invocation
- **object** (NSObject *)—the receiver of the –addObserver…invocation
- **change** (NSDictionary *)—all of the information about what changed
- **context** (void *)—the same value passed in the –addObserver…invocation



Key Value Observing

–[NSObject observeValueForKeyPath:ofObject: change:context:]

- **change** (NSDictionary *)
  - Information about what changed
  - Possible entries:

```
NSKeyValueChangeKindKey
NSKeyValueChangeIndexesKey
NSKeyValueChangeNewKey
NSKeyValueChangeOldKey
```

Key Value Observing

–[NSObject observeValueForKeyPath:ofObject: change:context:]

- **NSKeyValueChangeKindKey** (NSNumber *)
  - What kind of change it was

```
NSKeyValueChangeSetting
```

The value of an attribute or to-one relationship has been changed; e.g., someone invoked –setValue:forKey:

```
NSKeyValueChangeInsertion
NSKeyValueChangeRemoval
NSKeyValueChangeReplacement
```

A to-many relationship has been changed; e.g., someone invoked –mutableArrayValueForKey: then mutated the result

Key Value Observing

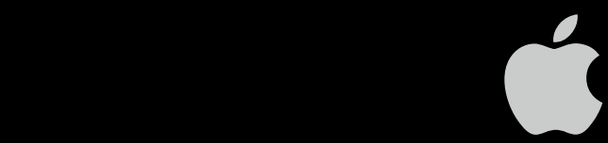
–[NSObject observeValueForKeyPath:ofObject: change:context:]

- **NSKeyValueChangeIndexesKey** (NSIndexSet *)
  - The indexes of the inserted/removed/replaced objects
  - Not present for NSKeyValueChangeSetting (not applicable)

Key Value Observing

–[NSObject observeValueForKeyPath:ofObject: change:context:]

- **NSKeyValueChangeOldKey** and **NSKeyValueChangeNewKey**
  - Only present if requested using the “old” and “new” options at -addObserver…time
  - For NSKeyValueChangeSetting, the old and new values of the attribute or to-one relationship
  - For NSKeyValueChangeInsertion, etc., arrays of objects that were inserted/removed/replaced



Key Value Observer Notification

Manual KVO-compliance

- Before any change to an attribute or to-one relationship invoke –[NSObject willChangeValueForKey:]
- After the change invoke –[NSObject didChangeValueForKey:]

```
@implementation SKTGraphic

- (void)setStrokeLineWidth:(float)width {
    [self willChangeValueForKey:@”strokeLineWidth”];
    ...do the actual change...
    [self didChangeValueForKey:@”strokeLineWidth”];
}

@end
```

Key Value Observer Notification

Manual KVO-compliance

- Before any change to a to-many relationship invoke
  –[NSObject willChange:valuesAtIndexes:forKey:]
- After the change invoke
  –[NSObject didChange:valuesAtIndexes:forKey:]

```
@implementation SKTDrawDocument
- (void)removeGraphicsAtIndexes:(NSIndexSet *)indexes {
  [self willChange:NSKeyValueChangeRemoval
    valuesAtIndexes:indexes forKey:@”graphics”];
  ...do the actual removal here...
  [self didChange:NSKeyValueChangeRemoval
    valuesAtIndexes:indexes forKey:@”graphics”];
}
```

Key Value Observer Notification

Automatic KVO-compliance

- Controlled by +[NSObject automaticallyNotifiesObserversForKey:], which returns YES by default
- All invocations of KVC methods result in observer notifications
- All invocations of KVC-*compliant* methods result in observer notifications

Key Value Observer Notification

Automatic KVO-compliance

- All –setValue:forKey: messages cause observer notifications for that key
- So do –set*Key*: messages!
- All mutations of the result of –mutableArrayValueForKey: cause notifications for that key
- So do –insertObject:in*Key*AtIndex: and –removeObjectFrom*Key*AtIndex: messages!

Key Value Observer Notification

Automatic KVO-compliance

- You don’t have to sprinkle –willChangeValueForKey:/–didChangeValueForKey: all over your code
- Access values consistently through KVC-compliant methods, and we do the rest
- Manual notification always available for when that’s not feasible

Key Value Binding (KVB)

- New API
- Ties it all together

Key Value Binding

What does it do?

- Allows the value of an object's property to be tied to the value of another object's property
- Provides plenty of flexibility; objects that support binding can provide custom options
- Uses key value coding and key value observing

Key Value Binding

What happens at nib-loading time

- Named properties of views and controllers are bound to properties in controllers and model objects
- Binding implies observing using KVO
- Binding also implies updating of bound-to values in response to user actions using KVC
- AppKit classes that support binding know what to do with observer notifications
- They know how to use –setValue:forKey: too

Key Value Binding

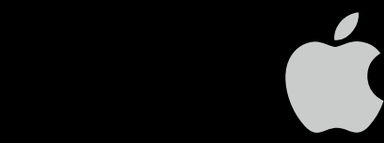
–[NSObject bind:toObject:withKeyPath:options:]

- Sent at nib-loading time to views and controllers for each binding you create in IB
- Can be invoked manually
- There's an –unbind: method too

Key Value Binding

–[NSObject bind:toObject:withKeyPath:options:]

- **bindingName** (NSString *)
  - Identifies a property whose value can be bound to the value of a property in another object
  - For example, “value,” “editable,” “textColor”
- **observableObject** (NSObject *)
  - The object containing the bound-to property
- **keyPath** (NSString *)
  - The key path that identifies the bound-to property
- **options** (NSDictionary *)



Key Value Binding

–[NSObject bind:toObject:withKeyPath:options:]

- **options** (NSDictionary *)
  - The parameters you can set in IB's binding inspector
    - Value transformer
    - Placeholders
    - Options that are specific to the binding

Controller Classes

- NSController
  - NSArrayController
  - NSObjectController
  - NSUserDefaultsController

NSArrayController

- Bind a table view to an array controller's "arrangedObjects"
- Bind the array controller's "arrayRelationship" to a to–many relationship of a model object
  - KVO notifications from the model object cause KVO notifications to the table view, taking sorting into account
  - Selection is kept valid
- Or, use the –setContent: method and let it manage the lifecycle of an array of objects

NSArrayController
Selection Management

- Bind the “selectionIndexes” of a table view to an array controller’s “selectionIndexes”
- Bind the properties of UI controls to “selection.*modelKey*”

```
[aColorWell bind:@”value”
  toObject:aGraphicsArrayController
  withKeyPath:@”selection.fillColor”
  options:optionsDictionary];
```

- Changes to selectionIndexes cause KVO notifications for every observed selection.*modelKey*

NSArrayController
Selection Management

- Built-in support for user-friendly behavior

```
- (void)setAvoidsEmptySelection:(BOOL)flag
- (void)setPreservesSelection:(BOOL)flag
- (void)setSelectsInsertedObjects:(BOOL)flag
```

- Programmatic control over selection

```
- (void)setSelectionIndexes:(NSIndexSet *)indexes
- (void)addSelectionIndexes:(NSIndexSet *)indexes
- (void)removeSelectionIndexes:(NSIndexSet *)indexes
```

NSArrayController

Sorting

- Bind the “sortDescriptors” of table view to the “sortDescriptors” of an array controller
- Changes to sortDescriptors cause KVO notifications for arrangedObjects

NSObjectController

- Bind the properties of UI controls to "selection.*modelKey*" of an object controller
- Bind the object controller's "objectRelationship" to a to–one relationship of a model object (or another controller)
  - Handles complicated situations like null values and not-always-applicable KVC properties cleanly
    - NSNoSelectionMarker
    - NSNotApplicableMarker
- Or, use the –setContent: method and let it manage the lifecycle of the object

NSUserDefaultsController

- Bind the values of UI controls to "value.*preferencesKey*" of a user defaults controller
- Gives easy access to up-to-date preferences values via KVO to other parts of the application
- Makes it easy to specify initial values for preferences

```
- (void)setInitialValues:(NSDictionary *)initialValues;
```

- +sharedUserDefaultsController is easy to get to in IB

Demo

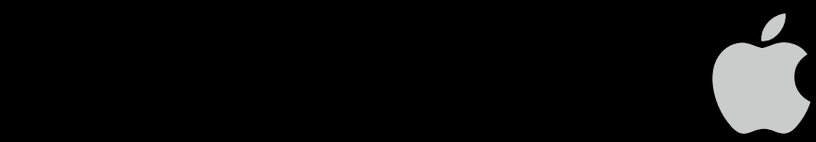
Performance

- Benefits of the new technology
- KVC performance advice
- KVO performance advice

Performance

Benefits of Key Value Observing

- Efficient observation info storage
  - Observation info is shared whenever possible
- Low overhead added to Key Value Coding
- Good default implementations of everything, but open to optimization for specific cases



Performance

Efficient KVC-compliance

- Don’t override KVC methods like –valueForKey: gratuitously
  - Prevents KVC from caching Objective-C runtime info

Performance

Efficient KVC-compliance

- Some *optional* KVC-compliant methods for to-many relationships I haven't told you about yet
- If you implement

```
- (unsigned int)countOfKey;
- (id)objectInKeyAtIndex:(unsigned int)index;
```

- And measuring says it's worth it, then also implement

```
- (void)getKey:(id *)objectBuffer
    range:(NSRange)rangeObjectsToGet;
```

Performance

Efficient KVC-compliance

- Likewise for mutable to-many relationships
- You have to implement

```
- (void)insertObject:(id)object
    inKeyAtIndex:(unsigned int)index;
- (void)removeObjectFromKeyAtIndex:(unsigned int)index;
```

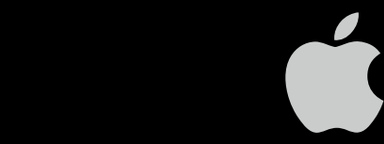
- You can also implement

```
- (void)replaceObjectInKeyAtIndex:(unsigned int)index
    withObject:(id)object;
```

Performance

Efficient KVC-compliance

- We'll add more KVC optimization opportunities in the future
  - More optional KVC-compliant methods for to-many relationships
  - More methods that take ranges? NSIndexSets? We'll see
- We don't intend to change the list of methods required for KVC-compliance



Performance

Efficient KVC usage

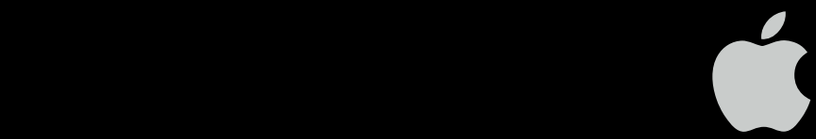
- Don’t invoke –valueForKey: and –mutableArrayValueForKey: redundantly
  - Overhead is reasonable, but more than a simple Objective-C message dispatch
- Try to use value arrays just like any other array
  - Specific optimizations should go in model code

Performance
Efficient KVO-compliance

- Override –setObservationInfo: and –observationInfo if measurement indicates it's appropriate

```
@implementation OneOfMany
- (void)setObservationInfo:(void *)observationInfo {
  _observationInfo = observationInfo;
}
- (void *)observationInfo {
  return _observationInfo;
}
@end
```

- Take advantage of the fine notification granularity that's possible



Performance

Efficient KVO usage

- Don’t observe unnecessarily
  - AppKit views only observe bound-to properties
  - Table views only observe for visible rows

Performance
Efficient KVO usage

- One more pair of methods I haven't told you about

```
@interface NSArray(NSKeyValueObserverRegistration)
- (void)addObserver:(NSObject *)observer
    toObjectsAtIndexes:(NSIndexSet *)indexes
    forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options
    context:(void *)context;
- (void)removeObserver:(NSObject *)observer
    fromObjectsAtIndexes:(NSIndexSet *)indexes
    forKeyPath:(NSString *)keyPath;
@end
```

- In common cases they boil down to –setObservationInfo: messages sent to each object

Wrap Up

Roadmap

400 Introduction to Developing Applications with Cocoa	Nob Hill Mon., 5:00 pm
410 Cocoa Update	Presidio Wed., 10:30 am
412 What's New in Cocoa User Interface Programming	Presidio Wed., 3:30 pm
413 Cocoa 2D Drawing Techniques	Nob Hill Wed., 5:00 pm

Roadmap

424 Cocoa Tips and Tricks	Mission Fri., 2:00 pm
427 What's New in Cocoa Text	Presidio Fri., 5:00 pm

Who to Contact

John Geleynse
Manager, Software Evangelism
geleynse@apple.com

Cocoa Feedback
For suggestions and feedback
cocoa-feedback@group.apple.com

Developer Technical Support
Code-level questions (fee-based)
dts@apple.com

http://developer.apple.com/wwdc2003/urls.html

For More Information

Reference Library

Documentation

Forthcoming—check headers

/System/Library/Frameworks/Foundation.framework/Headers
/System/Library/Frameworks/AppKit.framework/Headers

Q&A

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference